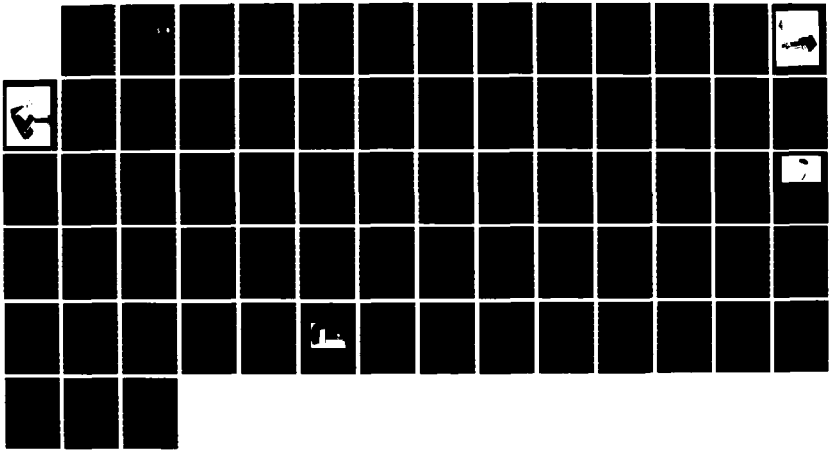
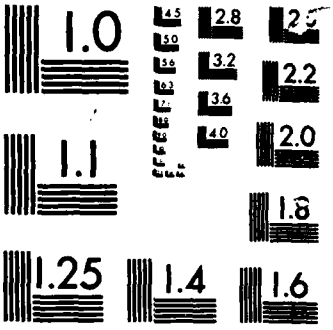
AD-A168 499 DEVELOPMENT SYSTEM FOR FLEXIBLE ASSEMBLY SYSTEM(U) SRI 1/1
INTERNATIONAL MENLO PARK CA R C SMITH FEB 86
AFOSR-TR-86-0329 F49620-84-K-0007

UNCLASSIFIED F/G 13/9 NL





MICROCOPY CHART

SRI International

AFOSR-TR- 86-0329

DEVELOPMENT SYSTEM FOR FLEXIBLE ASSEMBLY SYSTEM

Annual Report

Covering period 1 February 1984 through 31 January 1985.

By: Randall C. Smith
Computer Scientist
Robotics Laboratory

Prepared for:

Air Force Office of Scientific Research
Bolling Air Force Base, Washington, D.C. 20332

and

Air Force Wright Aeronautical Laboratories
Materials Laboratory
Manufacturing Technology Division
Wright-Patterson Air Force Base, Ohio 45433

Attn: Ted Brandewie
AFWAL/MLTC

Contract F49620-84-K-0007

SRI Project 7239

SRI International

333 Ravenswood Ave • Menlo Park CA 94025
415 326-6200 • TWX 910-373-2046 • Telex 334-486

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS none
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Unlimited Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SRI Project 7239	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 86-0329
6a. NAME OF PERFORMING ORGANIZATION SRI International 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Air Force
6c. ADDRESS (City, State and ZIP Code) 333 Ravenswood Avenue Menlo Park, California 94025	7b. ADDRESS (City, State and ZIP Code) Office of Scientific Research Building 410 Bolling Air Force Base, DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U.S. Air Force 8b. OFFICE SYMBOL (If applicable) NE	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-84-K-0007
8c. ADDRESS (City, State and ZIP Code) Office of Scientific Research Building 410 Bolling Air Force Base, DC 20332	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. 2306 / TASK NO. A3 / WORK UNIT NO.
11. TITLE (Include Security Classification) Development System for Flexible Assembly System.	
12. PERSONAL AUTHOR(S) Randall C. Smith	
13a. TYPE OF REPORT Annual 13b. TIME COVERED FROM 2/1/84 TO 1/31/85	14. DATE OF REPORT (Yr., Mo., Day) February 1986 15. PAGE COUNT 68
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In the one year period from February 1, 1984 to January 31,1985 of the AFOSR sponsored project, Contract F49620-84-K-0007, SRI performed four research and development tasks: We have developed theoretical methods for improving the ability of a computer to estimate spatial relationships among objects when the relationships are uncertain; we have developed a very fast method for determining collisions between a robot and its environment in simulation, suitable for hardware implementation; we are developing a method for estimating the location of a workpiece in a manipulator's hand, by measuring forces and torques on the hand as it moves the object; and we are developing an interactive-graphic, off-line robot workcell programming, modeling, and simulation system, called WORKMATE. WORKMATE has been used to generate a demonstration pick and place task incorporationg vision, executed in our laboratory workcell

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Maj Hager	22b. TELEPHONE NUMBER (Include Area Code) 202-767-4932 22c. OFFICE SYMBOL AFOSR/NE

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

CONTENTS

ILLUSTRATIONS

I OBJECTIVE

The objective of this research is to investigate some selected basic problems in flexible assembly that make human programming of assembly tasks difficult and to incorporate the results of such investigation into a graphics-oriented, assembly task development system.

II AIR FORCE RELEVANCE

This research addresses some problems in flexible assembly of electromechanical components. Improved automation of small batch assemblies should raise production efficiency, improve product quality, and lower costs.

III THE DIFFICULTIES IN MANUAL PROGRAMMING

It is difficult to write programs for a flexible assembly system because of the sensing and decision making capabilities entailed. From previous experiences at SRI in sensor-guided assembly, we picked three problems we considered most important for research:

- Locational Uncertainty—Estimation of errors in relations among workpieces, sensors, and effectors due to part tolerances, measurement errors, and positioning errors.
- Spatial Reasoning—Analyzing the relationships among solid objects in a three-dimensional space.
- Sensor Usage—Selecting sensors, determining their parameters, estimating sensor output values, and verifying actions by sensing (execution monitoring).

The current status of our research in these areas is summarized below, with detailed information contained in three appended papers.

A. Location Uncertainty

Because of part tolerances, measurement errors, and positioning errors, the locations of objects and relevant features (such as holes) are never known exactly. Transformations, which represent relative translations and rotations between object coordinate frames, are thus in error. These transformations are often compounded—relating, for example, the coordinate system of a peg in a manipulator's hand to a hole in a workpiece. The magnitude of the total error in the relationship should be analyzed to ensure that the error falls within the accuracy requirement for successful completion of a given task (e.g., putting the peg in the hole).

A representation of spatial relationships which incorporates knowledge about uncertainties, in the form of probabilities, is described in an appended paper. Our initial research has concentrated on spatial relationships with three degrees of freedom (two translations and a rotation in the plane). Within that framework, we present a first-order method for estimating the error when uncertain relationships are

(1) Compounded, increasing the overall uncertainty

(2) Merged, or "averaged," reducing the uncertainty

Compounding is illustrated by the sequential motions of a mobile robot, whose every (noisy) move increases the uncertainty about its current location with reference to its starting point.

In merging, two estimates of a relationship are combined to produce a better estimate of the relationship. A (noisy) sensor measurement of the robot's location can be combined with a second estimate of the location obtained by compounding the movements, providing a better location estimate than either of the original pieces of information.

The first-order estimation method provides a general tool for combining

uncertain information (given in probabilistic form), and is quite accurate over the range of errors anticipated in assembly processes (or mobile robot applications). The work is an advance over previous work which used worst-case analyses of compounded errors, because our method, when applicable, produces much more accurate estimates. Accuracy of our estimates have been checked by Monte Carlo simulations.

The uncertainty analysis will utilimately be applied as an off-line decision aid for picking assembly strategies that can accommodate the estimated spatial variations in a particular situation.

B. Spatial Reasoning

A particularly important problem in spatial reasoning is the problem of collision avoidance—finding a safe trajectory for a manipulator through an environment of obstacles. The general problem is unsolved. Our work has focussed on developing tools that aid a user of an off-line robot programming system in defining robot trajectories that are collision free. Robot motions can be visually inspected by the programmer, if they are graphically simulated. However, experimentation showed that this kind of visual inspection for collisions in the simulated robot workcell was tedious and prone to error. An automatic technique to detect simulated collisions quickly was developed, and is described in a second appended paper. The technique relies on the use of VLSI "clipping" hardware, which will be common in advanced graphic workstations of the future. Such hardware exists in the IRIS 2400 graphic workstation used in our off-line programming system, called WORKMATE. The paper describes an algorithm that can detect collisions (in simulation) between a manipulator and its environment at high speed—sufficiently fast for animation. It is implemented in WORKMATE.

C. Sensor Usage

It is highly desirable to develop approaches for determining the location of a workpiece in a manipulator's hand; the error in the grasp can then be estimated and corrective motions made by the manipulator. A method is being developed to determine the error in the grasp of an object based on measured forces and torques exerted by the object on the robot's wrist. The mis-position of the object can be determined by "weighing" it after putting the hand in several different positions. The mis-orientation of the part in the hand can be determined by measuring forces and torques at the robot wrist during a controlled acceleration. Thus, if

(1) The velocity/acceleration of the robot motion can be specified by the programmer

(2) A sensitive force/torque sensor is placed on the robot's wrist.

then it is theoretically feasible to accurately estimate the error in the grasp of a held object, even while the manipulator acquires and transfers it. The advantage of this procedure lies in removing the necessity of first moving the part to an inspection area, where the grasping error would be determined—thus saving time. A major implementation problem is that robot manufacturers do not currently provide the user with velocity and acceleration control over the robot. However, rather than controlling the acceleration of the robot, we can define several fixed motions for the robot to make, and measure the accelerations. When these motions are later made by the manipulator, the acceleration parameter, estimated a priori, can be used in our calculations. This laboratory experiment has not yet been performed. Details of this ongoing work will be supplied in a paper appended to the final report.

D. Workmate

Using results derived from our research on the above basic problems, and existing methods, we are developing a workstation modeling, analysis, training, and emulation system (WORKMATE). WORKMATE will be described briefly, followed by an account of an off-line programming demonstration that used it. More detailed information is being prepared in paper form, and will be submitted with the final project report. A long-range goal of this effort is to implement a graphic-based workstation that a non-specialist can use to develop, simulate, and debug robot workcell programs off-line. Accordingly, with this philosophy, user interaction with WORKMATE will be through the use of:

(1) Graphics to define geometric information

(2) Menus to encapsulate system commands or options

(3) Use of an electronic mouse or joystick to point at graphic objects, or provide control information.

To the extent possible, WORKMATE will avoid forcing the user to enter numeric data, or write instructions in textual programming languages. WORKMATE is written in the C programming language, and is implemented on a Silicon Graphics IRIS 2400 graphic workstation.

1. *Modeling*

It is anticipated that ***at least*** two representations will be used in the geometric models of the workcell and its components: models that can be rendered quickly for interaction with the user; and detailed models when detail is more important than display speed. We believe that detailed models are best derived from a commercial modeling package, and thus the work entailed involves interfacing it to the WORKMATE system. We have deferred that action indefinitely, while developing a simpler modeling package which supports quick rendering, real-time interaction with the user, and animation. Many of our research problems can be addressed directly with the modeling capabilities so far developed.

The current modeling system defines objects as collections of convex volumes. Object surfaces are drawn—not just wire frames of the surfaces. The surfaces are colored and shaded, and hidden surfaces are removed. The models are currently defined in text files, and are parsed into a geometric data base when loaded by WORKMATE. The underlying facilities have been developed to enable interactive graphic definition of the models, but that extension has not yet been made.

To define a collection of objects, their relationships, as well as their surfaces must be defined. Relationships are defined between object coordinate frames, and there are six degrees of freedom (three translation and three rotation) which must be given fixed values. The relationship is represented as the product of a primitive translation matrix (4 x 4) and three rotation matrices (4 x 4) in homogeneous coordinate form. Thus

X = translate(x,y,z) * rot(z,phi) * rot(y,theta) * rot(x, psi)

represents a relative change by first translating x,y,z distances along the current axes, and defining the new orientation by successive "roll, pitch, and yaw" rotations.

a. Robot Modeling

The modeling system supports the notion of "joint," and allows any of these degrees of freedom to be labeled as variables, thus implementing rotational and prismatic jointed mechanisms.

The modeler can "build" a robot with any number of joints, by supplying the robot's joints, links, and similar parameters given directly from the "A-matrices" (Dennavit and Hartenberg matrices) defining the robot's kinematic structure. The modeler also specifies joint limits and maximum velocities as part of the joint definitions. The robot's "skin" is modeled as convex volumes defined with respect to the joint axes. Given this information in the modeling file,

WORKMATE can parse the robot definition, and create a graphically simulated robot. Two robots, the PUMA 560 (Figure 1), and a GMF S-480 robot* containing a parallelogram linkage (Figure 2) have been modeled using the same system. The user can access menus to position and move the robot by controlling the joints, and ask the system to simulate a generalized joint-interpolated motion of the robot through a series of saved poses. These operations use only forward kinematic calculations.

Some robot positioning commands in the menus require inverse kinematic calculations—i.e., solutions for the robot joint values given a desired Cartesian coordinate description of the robot tool position and orientation. For a general robot, there may not be a closed form solution to the inverse kinematics, and no attempt is made to analyze the robot structure in order to produce these calculations automatically. Instead, the modeler must supply a routine to perform the inverse kinematic calculations—producing joint values from a specified Cartesian tool position and orientation. A pointer to this routine is saved when the model file is parsed; if no inverse routine is specified, a default routine will print a warning message whenever menu commands requiring inverse calculations are accessed.

2. *Analysis*

Having modeled the workcell components, and configured a workcell with them, the next problem is to describe a task for the cell to perform. We have defined some simple, high-level tasks, such as PICKUP or FIND (defined for our binary vision system), and built a menu of these tasks to present to the user. The user decides which task should be performed next, and picks it from the menu. WORKMATE has a semantic model for each task, and uses it to prompt the user for necessary information. The user indicates some of this information by pointing, or "picking" an object in the graphic scene with the graphic cursor (e.g.,

---

*We gratefully acknowledge GMF for supplying the modeling information.

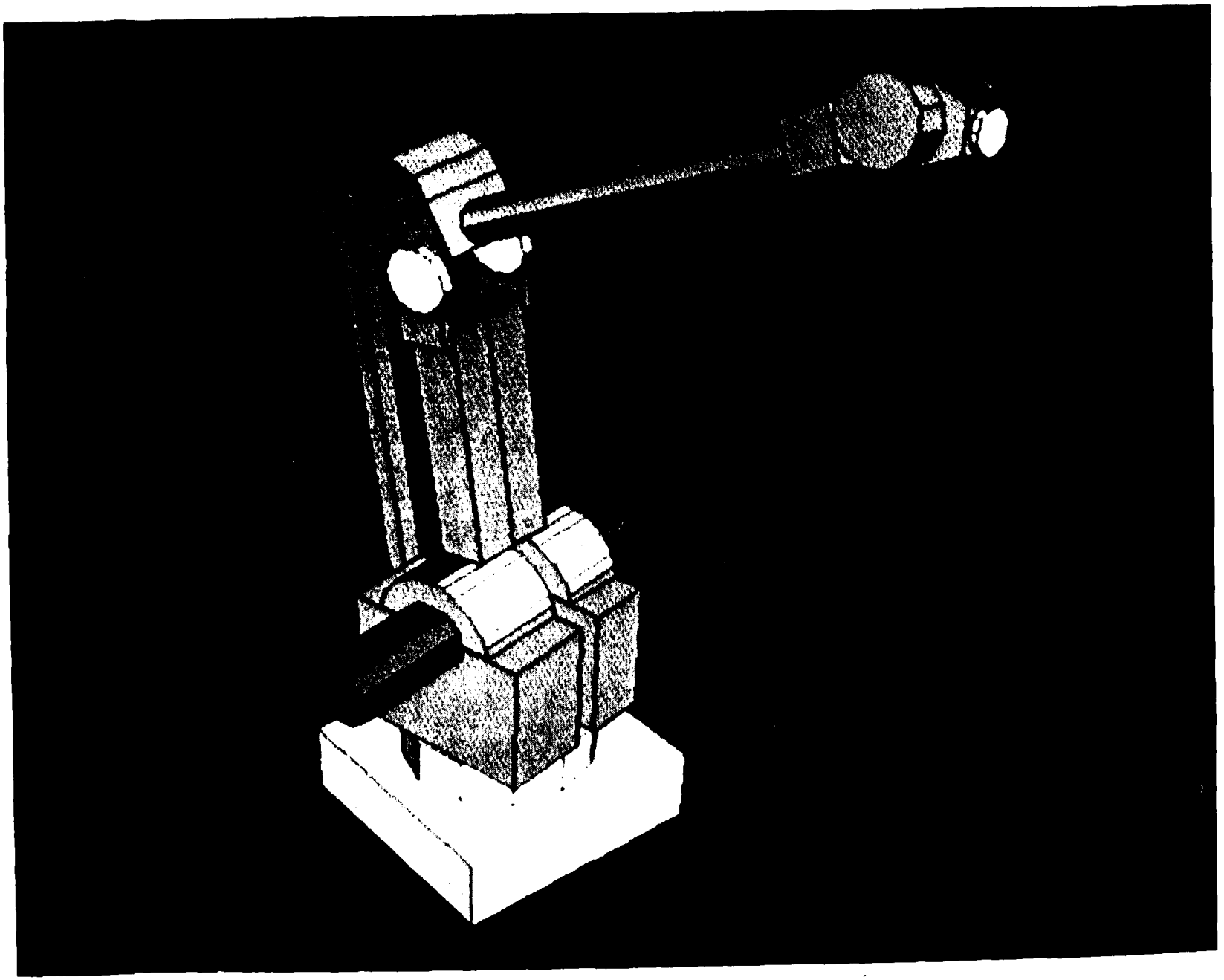


Figure 2: GMF S-480 Robot Model

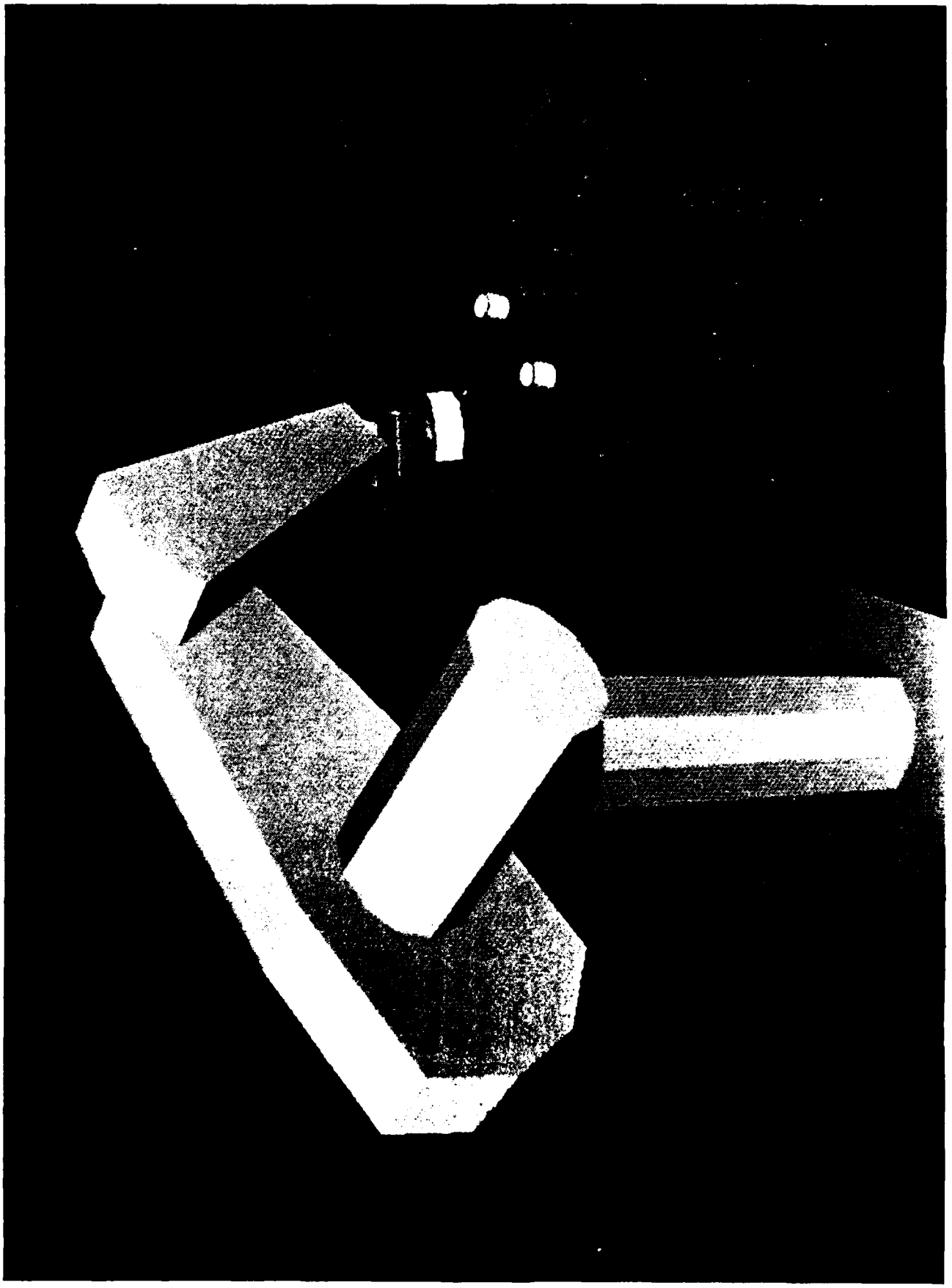


Figure 1: PUMA 560 Robot Model

answering the question, "Which robot should be used?"). Other information about desired geometric relationships between objects can be trained, as discussed in the following section.

To date, several sequential programs (those without loops or conditional branches) have been developed off-line by this method of "pasting-together" tasks from menu. Development of more elaborate programs is outside the scope of the current effort.

3. *Training*

While the user is defining the nominal task sequence, the analysis package determines what information must be supplied by the user to implement the tasks. The user typically supplies information by:

(1) Pointing at objects in the scene with a cursor

(2) Modifying system parameters by moving a joystick or mouse (e.g., robot joint values, positions and orientations of objects).

Several menus have been developed for teaching robot positions, or training robot paths. One menu allows individual joint control, and is presented graphically like the typical robot teach pendant. Other menus allow the user to move the robot tool in straight lines in tool coordinates, robot base coordinates, or the coordinates of any other designated object in the scene. A tool trajectory is defined as a series of static poses, and information about whether the trajectory between the poses is to be in straight lines, or joint interpolated. Additionally, the trajectory may be defined relative to a variable coordinate base frame, so the same relative motion can be made, for example, in acquiring a part from a location to be determined by sensing at task execution time.

To facilitate robot motion training, a menu of interactive view positioning commands was developed; these include panning, zooming in and out, centering the view on different objects, and changing the azimuth and elevation of the "eye."

To define motions of a manipulator's end-effector in space with a graphic interface, the user needs more than a single two-dimensional perspective view of the scene. We have experimented with simultaneously displayed top, front, and side views to provide more three-dimensional information. We implemented another viewing mode with a scene view, and a simultaneous view from an "eye in the hand" of the manipulator as it is maneuvered. We have also developed an anaglyphic (red/green) stereo viewing mode. Two pictures, one in red, the other in green, are presented to the viewer who wears glasses with the same colored filters. A striking perception of depth is created---elements of the scene can be made to "stand out" from the screen up to seven or eight inches. This viewing mode may be particularly useful when maneuvering the manipulator's tool into position near a workpiece.

4. *Emulation*

Once a portion of a task sequence has been specified, and its parameters given values, the user will want to view the resulting program fragment in *simulation, and get other simulation* data. WORKMATE simulates the actions of robots in the workcell and draws a scene after each simulation time increment. The time increment is under control of the user, and can be dynamically altered simply by moving the mouse over a graphic "time line." Thus, time can be speeded up, slowed down, or stopped at any arbitrary point in the simulation. Though not a current capability, it should not be a problem to run sections of the simulation backwards in time as well. A practical result of this flexibility is that uncritical sections of simulation can be bypassed (manipulators just "jump" from their starting points to their destinations at the future time). Alternatively, critical simulation sections can be slowed or stopped, allowing the user to change viewpoints for better inspection of the action, occuring in slow motion.

5. *Demonstration*

WORKMATE was demonstrated in the off-line programming of a visually-guided part acquisition task. Prior to developing the off-line program, a real workcell was set up in the laboratory. It consisted of one Unimation PUMA 560 manipulator, a servoed two-fingered electric hand, and a worktable containing simple block objects. The objects were illuminated from above and viewed with a camera attached to an Automatix AV4 vision system. Two-dimensional silhouette images of the objects were captured, and used in building prototypes to be used in automatic recognition of the objects.

The robot and vision system activities are coordinated by commands sent to them from a VAX 11/750 over an Ethernet network. Programs generated off-line by WORKMATE produce code in a text file that is passed over the Ethernet to the VAX, and there interpreted for execution.

The model workcell configuration is calibrated by reading in special calibration data files—initially containing only the nominal relationships among all the components defined in the cell layout. After physical calibration of the equipment to determine the actual coordinate frame relations, the calibration files are updated. When the model workcell is next regenerated, the computed (calibrated) relationships are used; and thereafter, programs written off-line for this workcell should perform accurately on the real equipment.

The nominally configured workcell model was loaded into WORKMATE and an off-line program for calibrating the robot and camera to each other was written. It consisted of training a trajectory for the simulated arm, which positioned it at a number of points in the camera field of view. Next, a FIND command was programmed to request the vision system to locate (in image coordinates) the calibration object on the robot hand, when it stopped at each location. At execution time, the two sets of data—calibration object locations and associated image locations—were used to compute the relationship between the

robot and camera. This calibration information was then used to update the relational workcell model in WORKMATE, and used in a second off-line programming task.

The second task was to pick up a cylinder, located by the vision system, and place it accurately on top of a second cylinder, also found with vision. A series of actions to implement the task were selected from a task menu, including PICKUP, FIND, and MOVE. Once an action is selected from menu, the system guides the user through its development. For instance, when "Pickup" is selected, the user is asked to define graphically the object-relative approach, grasp, and departure motions of the robot. The motions are defined relative to the coordinate frame defined in a nominally positioned cylinder; when the real cylinder's location is found using vision during program execution, the trained robot motions will be made relative to that location.

A series of actions were thus picked from menu, and interactively defined to implement the task. The resulting task was graphically simulated, and finally, real workcell program code was generated by WORKMATE. The demonstration, including programming and execution, takes less than 30 minutes, and was presented to the Industrial Affiliates of the Robotics Laboratory during the last meeting in July 1985.

Appendix

DISTRIBUTION LIST

Appendix

DISTRIBUTION LIST

AFOSR

Lt. Col Harry V. Winsor
Acting Director
Electronic and Material Sciences
Air Force Office of Scientific Research
Bolling Air Force Base, DC 20332

Air Force

Ted Brandewie
Air Force Wright Aeronautical Laboratories
Materials Laboratory
Manufacturing Technology Division
Wright-Patterson Air Force Base, Ohio 45433

Lt. Michael F. Hitchcock
Project Leader
Manufacturing Sciences Program
Air Force Materials Laboratory (AFWAL/ML)
Wright-Patterson Air Force Base, Ohio 45433

Dr. Vincent J. Russo
Director
Manufacturing Science Program
Air Force Materials Laboratory (AFWAL/ML)
Wright-Patterson Air Force Base, Ohio 45433

DARPA

Dr. William E. Isler
Program Manager, Robotic Systems
Systems Sciences Division
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Dr. Clinton Kelly
Director, Defense Sciences Office
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Robert Rosenfeld
Defense Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, VA 22209

Army

Mr. Timothy Evans
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27701

Navy

LCDR Bart Everett
Special Assistant for Robotics
Naval Sea Systems Command, C90-M
Washington, DC 20362

Alan Meyrowitz
Office of Naval Research
Code 437
800 N. Quincy
Arlington, VA 22217

NSF

Mr. Norman Caplan
Program Manager, Elec., Comp., and System Eng.
National Science Foundation
1800 G. Street, N.W.
Washington, DC 20550

Howard Moraff
Program Director
Automation and Systems Interpertation
1800 G. Street, N.W.
Washington, DC 20550

Bernard Chern
National Science Foundation
1800 G. Street, N.W.
Washington, DC 20550

NBS

Dr. James Albus
Division Chief, Industrial Systems Division
Center for Manufacturing Engineering
National Bureau of Standards
Bldg. 220, Room A123
Washington, DC 20234

Stanford University

Dr. Thomas O. Binford
Department of Computer Science
Stanford University
Stanford, CA 94305

Dr. Robert H. Cannon
Stanford University
250 Durand Bldg.
Stanford, CA 94305

University of Michigan

Dean Daniel Atkins
University of Michigan
Chrysler Center
Ann Arbor, Michigan 48109

Brigham Young University

Del Allen
Brigham Young University
Provo, Utah 84602

SRI International

ON THE REPRESENTATION AND ESTIMATION OF SPATIAL UNCERTAINTY

Robotics Laboratory Technical Paper

September 1985

By: Randall C. Smith, Research Engineer
Robotics Laboratory

Peter Cheeseman, Senior Computer Scientist
Robotics Laboratory

SRI Projects 4760 and 7239

The work reported in this paper was supported by the National Science Foundation under Grant ECS-8200615 and the Air Force Office of Scientific Research under Contract F49620-84-K-0007

Submitted to: International Journal of Robotics Research

SRI International
333 Ravenswood Ave • Menlo Park, CA 94025
415 326-6200 • TWX 910-373-2046 • Telex 334-486

On the Representation and Estimation of Spatial Uncertainty

Randall Smith

Peter Cheeseman *

SRI International
333 Ravenswood Avenue
Menlo Park, California 94025

Abstract

This paper describes a general method for estimating the nominal relationship and expected error (covariance) between coordinate frames representing the relative locations of objects. The frames may be known only indirectly through a series of spatial relationships, each with its associated error. These uncertain relationships (called uncertain transformations) can arise from diverse causes, including positioning errors, measurement errors, or tolerances in part dimensions. This estimation method can be used to answer such questions as whether a camera attached to a robot is likely to have a particular reference object in its field of view. The calculated estimates agree well with those from an independent Monte Carlo simulation. The method makes it possible to decide *in advance* whether an uncertain relationship is known accurately enough for some task and, if not, how much of an improvement in locational knowledge a proposed sensor will provide. The method presented can be generalized to six degrees of freedom, and provides a practical means of estimating the relationships (position and orientation) between objects, as well as the uncertainty associated with the relationship.

---

* Currently of NASA-Ames Research Center

On the Representation and Estimation of Spatial Uncertainty

1 Introduction

In many applications it is necessary to reason on the basis of inaccurate information about spatial relationships among objects. For example, a mobile robot needs to represent and reason about the approximate relationships between itself and other objects. In addition, the robot must be able to use sensor information to reduce locational uncertainty (in both position and orientation) to a degree sufficient for achieving particular tasks. This problem is complicated in practice because the location of one object relative to another may be known only indirectly through a sequence of relative frames of reference with uncertainty. In this paper, we present a method for explicitly representing and manipulating the uncertainty associated with these transformations; we also show how sensors can be used to reduce this uncertainty. This formalism makes it possible to estimate, *in advance*, the answers to such questions as the probability of a robot going through a door given the current uncertainty of the robot and door location, the probability of a camera having an object in its field of view, whether a particular sensor will have sufficient accuracy to accomplish a particular task, and so on.

Brooks (1985) argues that it is not appropriate for mobile robots to use a global reference frame—a set of local reference frames linked via uncertain transformations is better. We show how the uncertainty of a frame relative to another can be estimated and how the reduction in sensing uncertainty can be mapped into any frame, regardless of in which frame the sensing was performed (e.g. the robot frame). Because of this flexibility, no particular frame is necessary as an absolute reference.

In the following sections, we show how to estimate the uncertainty in three degrees of freedom—(x, y, θ)—using a mobile robot as the example. As the robot moves from

one place to another, its uncertainty about its location with respect to its initial location grows. We present a method for making quantitative estimates of the resulting error, provided the moves are discrete. By using sensors (e.g. acoustic range, vision), the uncertainty of the location of sensed objects as well as that of the robot can be reduced. Because the estimation and update formulas and associated procedures require only simple matrix computations, the proposed method is computationally simple and fast. The theory assumes that the contributing errors are "small", allowing a good approximation with a first-order model of errors, and that the sensor errors are independent of the locational error. Although the example presented is for a mobile robot, the formulas for estimating locational uncertainty are applicable to many other domains. In fact, this work was originally motivated by the need to represent and reason about uncertainty in off-line programming applications for industrial manipulators.

2 Basic Operations for Estimating Uncertainty

In this paper, we introduce two basic operations that allow the estimation of the uncertain relationship between any two coordinate frames, given the chain of relative transformations linking them. These relative transformations (here called "uncertain transformations") typically arise from relative motions or from sensing operations. The first operation, called "compounding", allows a chain of relative uncertain transformations to be collapsed (recursively) into a single uncertain transformation. The final compounded transformation has greater uncertainty than its components. The second operation, called "merging", combines information from parallel uncertain transformations to produce a resultant uncertain transformation with uncertainty less than its components.

2.1 Compounding of Uncertain Transformations

A simple robot example of compounding shown in Fig. 1, is similar to that discussed in Brooks (1985)

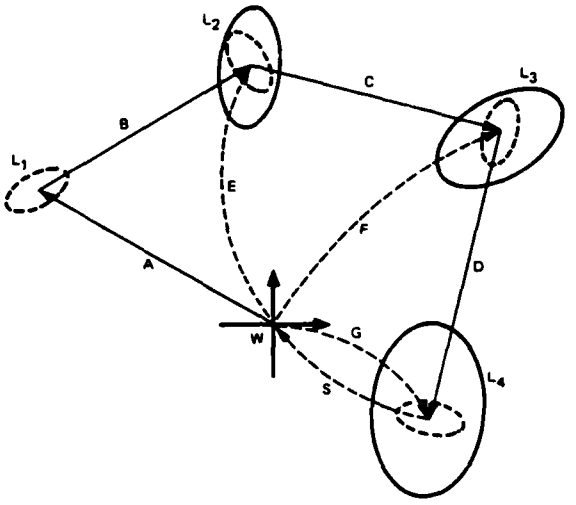


Figure 1—A Sequence of Uncertain Transformations

In this example, the robot makes a number of moves and ends up near its initial position W. The uncertainty of the robot's final location with respect to W is large, as indicated by the large error ellipse. The dashed error ellipses express the uncertainty of the robot with respect to its last position, while the solid ellipses express the uncertainty of the robot with respect to W. Note that the uncertainty of the robot's location with respect to the world frame grows with each move. In Section 3 we show how to calculate compounded uncertain transformations, such as E and G. An uncertain transformation consists of a nominal (mean) relation of a frame relative to another and a covariance matrix that expresses the uncertainty of the nominal transformation. This is represented symbolically by an arrow going from the reference frame to the relative location, as shown in Fig. 1. The diagonal terms of the covariance matrix

express the variance of the corresponding coordinate, and the off-diagonal terms give the covariances between these coordinates.

2.2 Merging of Uncertain Transformations

A different situation arises when there is more than one independent uncertain transformation relating the frame of interest relative to a given base. For example, if the robot in Fig. 1 observes its starting point (the frame W) from L_4, then we have two uncertain transformations from W to L_4—uncertain transformation G, calculated by compounding, and the inverse of uncertain transformation S given by the sensor. These two transformations are combined using a Kalman filter approach described in Section 4, to give a more accurate uncertain transformation G' that expresses the best linear estimate of the location of L_4 with respect to W. The error (covariance) of the uncertain transformation S comes from the intrinsic error of the sensor, which is assumed to be unbiased (i.e. the error distribution has a zero expectation around the nominal value).

The basic theory for calculating compound uncertain transformations is given in Section 3, and the theory for merging uncertain transformations is given in Section 4.

3 Estimating Compound Uncertain Transformations

In this section, we describe the procedure, called "compounding", for calculating the nominal location and associated error (expressed as a covariance matrix) of any object relative to any other object linked through a chain of "uncertain" transformations (e.g. Fig. 1). This approach differs from that presented by Brooks (1982,1985), who used a max/min representation of the error. The max/min approach assumes the worst case when errors are compounded and so can badly overestimate the error when the results are computed through several transformations. Also, the interpretation of the max/min bounds are unclear—Do they mean that no observation could ever

fall outside the bounds, or just very unlikely to do so (if so, how unlikely)? The approach described by Chatila and Laumond (1985) for the HILARE robot is more similar to that described here; however, they used a scalar error estimate of position, and were not concerned with angular error. Futhermore, they described a procedure that seems to back-propagate the results of measurements to update previous estimates. This procedure does not appear to be correct in general because the same uncertain relationships are reused in back-propagating, even though they are not new information.

3.1 Formulas for Compounding

In Fig. 1, we wish to describe the coordinates of L_2 (X_3, Y_3, θ_3) with respect to reference frame W. We are compounding transformations A (X_1, Y_1, θ_1) and B (X_2, Y_2, θ_2). The explicit transformation is given in Eqn. (1), and is derived from the formulas for transforming one frame into another, as shown in Paul (1981), for example.

$$\begin{aligned} X_3 &= f(X_1, Y_1, \theta_1, X_2, Y_2, \theta_2) \\ &= X_2 \cos\theta_1 - Y_2 \sin\theta_1 + X_1 \\ Y_3 &= g(X_1, Y_1, \theta_1, X_2, Y_2, \theta_2) \\ &= X_2 \sin\theta_1 + Y_2 \cos\theta_1 + Y_1 \\ \theta_3 &= h(X_1, Y_1, \theta_1, X_2, Y_2, \theta_2) \\ &= \theta_1 + \theta_2 \end{aligned} \tag{1}$$

We wish to estimate the means and covariances of these three functions. The variables are now assumed to be ***random variables***. The functions are approximated by a first-order Taylor series expansion about the means of the variables. The mean values of the functions will be the functions themselves [e.g., $\hat{X_3} \cong f(\hat{X_1}, \hat{Y_1}, \hat{\theta_1}, \hat{X_2}, \hat{Y_2}, \hat{\theta_2})$] (to first order). In addition to estimating the mean transformation, an uncertain transformation includes the associated covariance matrix **C** of this transformation. To estimate

the covariance matrix for this case, we express the previous Taylor series expansion in matrix form, resulting in the following (deviate) matrix:

$$\begin{pmatrix} \Delta X_3 \\ \Delta Y_3 \\ \Delta \theta_3 \end{pmatrix} \cong \mathbf{J}\,(\Delta X_1, \Delta Y_1, \Delta \theta_1, \Delta X_2, \Delta Y_2, \Delta \theta_2)^t \tag{2}$$

where $\mathbf{J}$ is the (3x6) Jacobian of the transformation evaluated at the mean values of the input variables:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial X_1} & \frac{\partial f}{\partial Y_1} & \frac{\partial f}{\partial \theta_1} & \frac{\partial f}{\partial X_2} & \frac{\partial f}{\partial Y_2} & \frac{\partial f}{\partial \theta_2} \\ \frac{\partial g}{\partial X_1} & \frac{\partial g}{\partial Y_1} & \frac{\partial g}{\partial \theta_1} & \frac{\partial g}{\partial X_2} & \frac{\partial g}{\partial Y_2} & \frac{\partial g}{\partial \theta_2} \\ \frac{\partial h}{\partial X_1} & \frac{\partial h}{\partial Y_1} & \frac{\partial h}{\partial \theta_1} & \frac{\partial h}{\partial X_2} & \frac{\partial h}{\partial Y_2} & \frac{\partial h}{\partial \theta_2} \end{pmatrix} = \begin{pmatrix} 1 & 0 & -(Y_3 - Y_1) & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & 1 & (X_3 - X_1) & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

and X_3, Y_3 are given in Eqn. (1). Covariance is defined as the expectation of the squared deviates. We "square" both sides of Eqn. (2) by multiplying both sides of the equation by their respective transposes. Taking the expectation of the result, we get the covariance matrix C_3:

$$\mathbf{C}_3 \cong \mathbf{J} \begin{pmatrix} \mathbf{C}_1 & 0 \\ 0 & \mathbf{C}_2 \end{pmatrix} \mathbf{J}^t \tag{3}$$

The 3x3 matrix $\mathbf{C}_3$ expresses the covariances of the coordinates of L_2 with respect to W, computed from the input covariance matrices $\mathbf{C}_1$ and $\mathbf{C}_2$ (which express the error on the input variables of A and B). Because an element of the covariance matrix is by definition $\mathbf{C}_{ij} = E(\Delta x_i \Delta x_j)$ and the standard deviation of variable x is $\sigma_x = \sqrt{E(\Delta x^2)}$, an element of the covariance matrix can be expressed as:

$$\mathbf{C}_{ij} = \rho_{ij}\sigma_i\sigma_j \quad ; \qquad \rho_{ij} = \frac{E(\Delta x_i \Delta x_j)}{\sqrt{E(\Delta x_i{}^2)E(\Delta x_j{}^2)}}$$

where ρ_{ij} is the correlation coefficient. On the diagonal of the covariance matrix, $i = j$, thus ρ_{ii} is 1, and $\mathbf{C}_{ii}$ is just the variance.

Equations (1) and (3) are all that is needed to compute the compound uncertain transformation for more than two component uncertain transformations. The method

consists of computing the compound of two adjacent uncertain transformations and effectively replacing the pair by this result. The uncertain transformation between any pair of connected frames can be computed by a combination of such reductions. That is, a series of relative transformations can be computed by compounding the first two transformations to form a new composite transformation, then compounding this new transformation with the third transformation to form a new composite, and so on. In Fig. 1, for example, **A** and **B** are compounded to give uncertain transformation **E**, then **E** and **C** are compounded to give **F**, and finally **F** and **D** are compounded to give **G**.

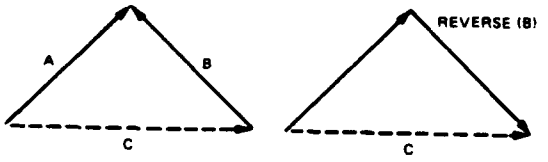


Figure 2(a) Figure 2(b)
Examples of Compounding Situations

A given directed graph of uncertain relationships may not be directly in a form ready to be computed using the above formulas. For example in Fig. 2(a), **B** is pointing in the opposite direction to the form assumed above. To create the correct form, it is necessary to first reverse **B** as in Fig. 2(b). This reversal is easily accomplished using the following formulas (the nominal inverse relationship):

$$\begin{aligned} X' &= -X\cos\theta - Y\sin\theta \\ Y' &= X\sin\theta - Y\cos\theta \\ \theta' &= -\theta \end{aligned}$$

As before, the first-order estimate of the mean values of the dashed coordinates is simply the given functions of the mean variables. The covariance matrix for the reversed transformation is computed from the given covariance matrix of **B**, (the matrix **C**):

$$\mathbf{C}' \cong \mathbf{J} * \mathbf{C} * \mathbf{J}^t$$

where **J** is the Jacobian of the above transformation equations; that is, **J** is given by:

$$\mathbf{J} = \begin{pmatrix} \frac{\partial X'}{\partial X} & \frac{\partial X'}{\partial Y} & \frac{\partial X'}{\partial \theta} \\ \frac{\partial Y'}{\partial X} & \frac{\partial Y'}{\partial Y} & \frac{\partial Y'}{\partial \theta} \\ \frac{\partial \theta'}{\partial X} & \frac{\partial \theta'}{\partial Y} & \frac{\partial \theta'}{\partial \theta} \end{pmatrix} = \begin{pmatrix} -\cos\theta & -\sin\theta & Y' \\ \sin\theta & -\cos\theta & -X' \\ 0 & 0 & -1 \end{pmatrix} \quad (4)$$

With this inverse operation and the pairwise compounding operation given above, it is possible to compute the compound uncertain transformation between any two frames of reference that are linked by a chain of uncertain transformations, as in Fig. 1.

3.2 The Assumptions for Compounding

The assumptions behind this compounding operation are:

- The first order approximation expressed by Eqn. (2) is sufficiently accurate.
- The covariances of different transformations are independent.

The first-order approximation is reasonable provided that the standard deviations of the variables (e.g. σx_1) are small, because we are neglecting terms of order $(\sigma_x)^2$ and higher orders. More accurately, the function should be "smooth" about the estimated point over an interval roughly the magnitude of a standard deviation of the variable. This approximation can underestimate the error for a covariance element when the corresponding Jacobian element is zero because, in this case, the approximation predicts that the function variance does not depend on the error in the corresponding input variable. In such cases, the second-order terms may be significant.

The assumption that successive uncertain transformations have independent error gives a compound covariance matrix with zero elements, as in Eqn. (3). If these errors are known to be dependent, a compound uncertain transformation with non-zero covariance elements can be given directly; that is, the dependent case can be handled by treating the dependent combination as a unit, with given compound covariance, and replacing the components with this unit.

4 Merging of Uncertain Transformations

The second basic operation is the combination ("merging") of two or more parallel uncertain transformations to obtain a more accurate combination. For example, in Fig. 1, we can compute **G**, which expresses the uncertain relationship of L_4 with respect to the frame W, and we are given the uncertain transformation S describing W with respect to L_4. The problem we are considering is how to estimate the relation of L_4 with respect to W given both parallel transformations. The procedure we use for merging is based on the use of Kalman filter theory (for static-state estimation).

4.1 Formulas for Merging

The first step is to find all the uncertain transformations linking the frames of interest. For each independent chain of transformations between these frames, compute the compound uncertain transformations. For the example in Fig. 1, **G** must be computed using the methods described in Section 3 to calculate the uncertain transformation of L_4 with respect to W. The next step is to ensure that all the parallel uncertain transformations to be merged are pointing in the desired direction, and to reverse those that are not. In the Fig. 1 example, **S** should be reversed. The reversal of an uncertain transformation is given by Eqn. (4) above.

Once all the parallel uncertain transformations to be merged have been computed (including any necessary reversals), the merging procedure combines them in pairs,

using the result of the last merge as an input to the next merge. The merging begins with any pair, and proceeds by merging one uncertain transformation at a time until all have been merged. Consequently, it is only necessary to describe the pairwise merge procedure. In the following, let $\mathbf{C}_1$ and $\mathbf{C}_2$ be the covariance matrices of the uncertain transformations to be merged, and $\mathbf{C}_3$ be the covariance matrix of the resulting merged pair. Similarly, $\hat{X}_1$ and $\hat{X}_2$ are the estimated mean values (expressed as a vector) of the transformations to be merged, and $\hat{X}_3$ is the resulting estimated mean. The first step in the merging operation is to compute the Kalman gain factor defined by:

$$\mathbf{K} = \mathbf{C}_1 * [\mathbf{C}_1 + \mathbf{C}_2]^{-1} \tag{5}$$

This factor is used to compute the required merged covariance matrix:

$$\mathbf{C}_3 = \mathbf{C}_1 - \mathbf{K} * \mathbf{C}_1 \tag{6}$$

and the merged mean:

$$\hat{X}_3 = \hat{X}_1 + \mathbf{K} * (\hat{X}_2 - \hat{X}_1) \tag{7}$$

These formulas are sufficient for merging any number of parallel uncertain transformations. The apparent asymmetry of Eqns. (5), (6) and (7) is not real, because the same result is obtained regardless of which transformation is labeled $\mathbf{C}_1$ or $\mathbf{C}_2$, and so on. That is, the order in which the mergings are performed is irrelevant. In one dimension, these formulas reduce to the following simple forms (where V's are variances):

$$V_3 = \frac{V_1 V_2}{V_1 + V_2} \quad ; \quad X_3 = \frac{V_2}{V_1 + V_2} X_1 + \frac{V_1}{V_1 + V_2} X_2 \tag{8}$$

That is, the covariance matrices reduce to simple variances combined as shown and the merged mean is just a weighted average of the contributing means. If the compounding operation described in Section 3 is also reduced to one dimension, we find that variances in series transformations simply add to give the compound variance. If this result and Eqn. (8) are combined, we find an equivalence to electric circuit theory where variances

are analogous to resistances. That is, in a network of resistors, series resistances add and parallel resistances combine according to Eqn. (8) to give a combined resistance (variance) between any two points.

This equivalence points to a difficulty with the method described, because not all circuits can be analyzed by a simple combination of serial and parallel calculations (e.g. the Wheatstone bridge). That is, it is not always possible to "reduce" a network of uncertain relationships by replacing series or parallel components with their equivalent values. However, by deleting links in the uncertain transformation network, it is possible to produce a reducible network and so compute an estimate for the mean and covariance. By considering alternative deletions and finding the set that gives the "best" final covariance, it is usually possible to get an estimate that does not significantly overestimate the final covariance. Such estimates are overestimates because they do not utilize all the available information. Note that the first order approximation we are using can under or over estimate the error when the input errors are large, so the failure to use all the information may not be important.

4.2 The Assumptions for Merging

Kalman filter theory, or state estimation, is described in many texts on control theory; the derivation of the basic formulas is lengthy, and is not presented here (see Nahi, 1976). The basic assumptions of the theory in this context are:

- The covariance matrices to be merged are independent; that is, the errors in one uncertain transformation are not correlated in any way with the errors in any other uncertain transformation. This prevents the same sensor results being used more than once, for example.
- Covariance matrices contain only constants. This assumption prevents the elements of the covariance matrices from being dependent on the actual mean. If the covariance elements depend only weakly on the mean (compared with the

magnitude of the errors), no significant error is introduced by using covariances defined by evaluating the expression for the error at the nominal location.

- The errors are unbiased; that is, an input error (from which the covariance matrices are defined) has zero expectation around the mean value of the variable. This assumption excludes systematic errors, which are usually eliminated by suitable calibration.

- The covariance matrices and means to be merged are expressed in the same coordinate system (e.g. Cartesian). If this is not true, suitable mappings must first be performed.

When these assumptions are satisfied, the Kalman filter supplies the best linear unbiased estimate for the updated mean and variance matrices. Note that the Kalman filter theory does not require that the error model of the locational and sensor uncertainty be Gaussian. The Kalman filter theory can be extended in various ways if some of these assumptions are violated. For example, if the contributing covariances are not independent, the resultant system may be modeled by introducing an additional correlation matrix that corrects for the known dependencies.

5 A Mobile Robot Example

A mobile robot needs to use its sensors to build and update a world map to navigate in both known and unknown environments. At any stage, this map can be viewed as a network of uncertain relations that can be used to decide important navigational questions. Initially, the robot will take its starting position as the "world" frame. As it moves from its initial position, the uncertainty of its location with respect to the world grows with each successive move, as shown in Fig. 1. The error in these moves is usually given relative to the last location (e.g. wheel counters) and is represented by an uncertain transformation. After a number of such moves the robot's location with

respect to the world frame becomes so uncertain that the robot is unlikely to succeed in actions, (e.g. going through a doorway), based purely on its current information. The procedures described above allow the robot to estimate the uncertainty of its location and decide whether the accuracy given by this estimate is sufficient for the particular task. Note that because this estimate can be made ahead of time, the robot can decide that proposed motions will create too much uncertainty *before* they are performed, or that sensing will be necessary.

5.1 Robot Sensing

A mobile robot is usually equipped with sensors that allow it to determine the location of objects to an accuracy determined by the sensor resolution. Such sensed relationships are also represented as uncertain transformations in the uncertain transformation network, along with those due to motion. The sensed information allows the robot to reduce the uncertainty of its location with respect to other objects or the world. Because the reduction of uncertainty through sensing can be calculated ahead of time, it is possible to decide if proposed sensing steps will provide sufficient accuracy for a task *before* performing the sensing. If a proposed sensing step is not sufficiently accurate, alternative sensing strategies can be evaluated. This means that the uncertain transformation estimation procedure can be used for off-line programming of a robot, as well as for maintaining the best current "run-time" estimate.

It is important that the robot sense as accurately as practical the location of many reference objects while it is still in its initial position ("world"). Once the robot moves, these objects allow the robot to estimate its position with respect to the world with the greatest possible accuracy. Even if the robot is unable to sense the original reference objects, accurate world locations can still be found if the original reference objects are related to other observable objects through accurately sensed relationships. That is, a rich connected network of uncertain transformations allows the robot to locate itself with respect to any object (in the network) with an accuracy largely determined by

the accuracy of the sensors. Note that the method advocated here is for the robot to maintain a network of raw uncertain transformations resulting from individual motions or sensing operations and to compute composite uncertain transformations as required. This is different from the method proposed by Chatila and Laumond (1985), for example, who seem to update previously computed composite uncertainty relationships whenever new information becomes available ("backpropagation"). Update procedures can lead to incorrect results because they use the same uncertain relations more than once (the prior uncertain relations), even though there is no new information about them.

5.2 The Sensing Procedure

Given that it has decided to perform a sensor step, the procedure the robot should follow in adding a sensor based uncertain transformation to the current network is as follows:

1. Determine whether sensing is possible—Using the current estimated uncertain transformation calculated by the procedure described above, decide whether the sensor is capable of making the desired observations. For example, decide whether a wall is likely to be close enough to be detected with an acoustic range sensor, or whether the object to be sensed is in the field of view of a camera (an example of the latter case is given in Appendix A).

2. Decide whether the actual observation is reasonable—Given the uncertain prior location of the object to be sensed (estimated from the current uncertain transformations) and the error associated with the sensor, decide whether the probability of the actual observation is below a predetermined threshold. Such errors would occur, for example, if the camera locked onto the wrong object.

3. Combine multiple sensings—Merge independent measurements by the sensor into a single uncertain transformation using the merging Eqns. (5), (6) and (7). If the

sensing information is in a different coordinate system (e.g. Polar rather than Cartesian), then the merging operation is performed in the sensor system and the final uncertain transformation is mapped back into the Cartesian form (the specific formulas for performing this mapping in the case of Polar to Cartesian are given in Appendix B).

The above sensing procedure makes a number of assumptions:

- The error of the sensor is *independent* of the prior locational error.
- The error of the sensor is *independent* of the nominal location $\hat{X}$. This is equivalent to requiring that the sensor error be expressed by a constant covariance matrix (it could be a different constant matrix for measurements made from different positions). This requirement ensures that the update is linear, as required for this theory to be applicable. This requirement can be relaxed with little loss of accuracy provided that covariance is a slowly varying function of location compared to the error range.
- The sensor error is unbiased, i.e. the expected error is zero around the nominal value.

5.3 Monte Carlo Simulations

To test the assumptions in the theory, we ran an independent Monte Carlo simulation, where the robot's position is calculated many times using Gaussian distributions for the input errors. Fig. 3 shows the resulting robot positions as points with the estimated 90% confidence contour superimposed on them. This contour is formed by assuming that the underlying probability density function is Gaussian. Numerically, the estimated and simulated means and covariances generally agree to within 1%, unless the input errors are very large.

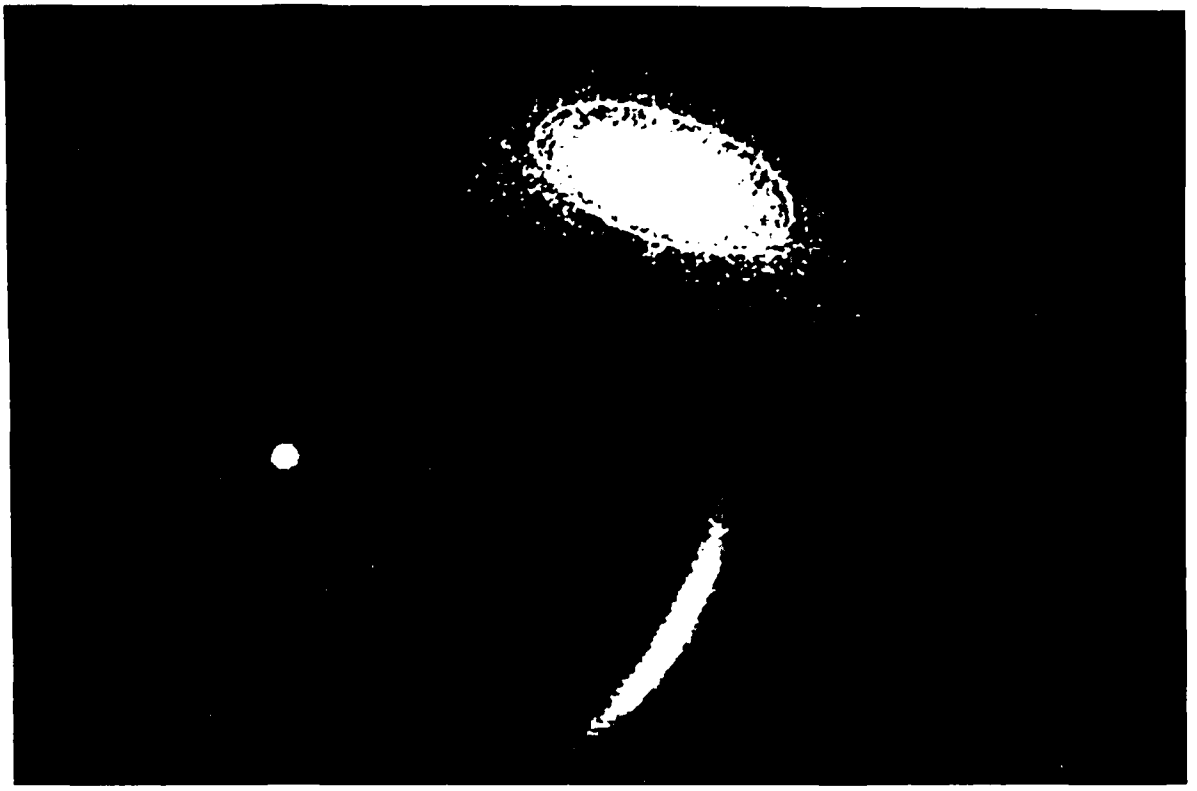


Figure 3—A Monte Carlo Simulation

If the angular errors are large, the probability of the actual position lying within a given error ellipse (assuming the distribution is Gaussian) diverges from the actual case (as in Fig. 3). However, the mean and covariance estimates still agree to within 1% — it is the Gaussian assumption that is breaking down. The divergence occurs because of *nonlinearities* in the Jacobian (i.e. its dependence on the trigonometric functions of θ) when the input angular errors are large. Even in this case, the probability distribution for the final position, formed from two approximately perpendicular motions, is very close to Gaussian.

When decisions requiring a probability distribution are necessary, such as deciding if an object is in a camera's field of view, it is necessary to assume a particular distribution, given only two moments (the mean and the variance). When the mean and variance of an unknown probability distribution are the only information available, a simple maximum entropy derivation gives the Gaussian distribution as the distribution that assumes the least information. The possibility of using higher order moments

being investigated. If the probability distributions of the input variables are not Gaussian, the central limit theorem assures us that the final compound distribution will be approximately Gaussian provided that there are a "large" number of inputs.

6 Discussion

The previous sections describe a procedure for making accurate quantitative estimates of the mean and covariance of the location (position and orientation) of any frame relative to any other given a network of uncertain transformations. If the distribution associated with these estimates is assumed to be Gaussian, then probabilities, such as whether a sensor can make a particular measurement (i.e. the expected location is within sensor constraints) can be calculated. Examples of the application of this theory are presented and the estimates are compared with Monte Carlo simulations for a three-degree-of-freedom mobile robot. The only significant divergence between the estimates and the simulation occurs when the angular error is large compared with a subsequent displacement error. Under these circumstances, the resultant probability distribution is significantly non-Gaussian (i.e. noticeably nonelliptical in Cartesian space). In all other cases, the mean and variance estimates agreed extremely well with the simulated values. Even when the angular errors are large, the nonlinearities tend to cancel, except when the moves are approximately colinear.

The formulas given for three degrees of freedom (X, Y, θ) can be easily extended to include the Z coordinate in the obvious way. Unfortunately, the extension to the full six-degrees-of-freedom case $(X, Y, Z, \theta, \phi, \psi)$ is not so simple. The main problem is the additional angular terms can introduce singularities ("poles") into the Jacobian; that is, the estimated covariance shows extreme sensitivity to the error of particular input angles in the vacinity of a pole. These singularities destroy the value of the estimates provided by the estimation procedure unless corrective measures are taken. One approach we are investigating is to prerotate the world frame so that none of the uncertain transformations have nominal angles in the vacinity of a pole.

There are many uses for the estimation procedures described in this paper. The major use motivating this work is the ability to estimate *ahead of time* when sensing steps are necessary and determining whether a particular sensor is capable of making a particular measurement or supplying the required accuracy. The procedure also supplies a quantitative method for judging when a sensor has "glitched"—i.e. its measured value is too unlikely, given the prior expectations. Another possibility being investigated is whether the results of running this procedure can be used to revise the error models used for particular mobile robots or sensors, giving a form of autocalibration.

The major limitation of the estimation procedure described in this paper is that only two moments (the mean and covariance) are estimated, hence the knowledge of the probability distribution satisfying these moments is very limited. By assuming the (multivariate) distribution is Gaussian, we can still make probabilistic estimates that agree with simulations very well, except when the angular errors are large. A way around this difficulty that we are investigating is to estimate a third moment (a "skewness" measure).

Appendix A: Derivation of Ellipse Parameters

The determination of whether the probability of observing the object is greater than a given threshold requires assuming that the probability distribution of our knowledge of the object's location is a multivariate (x,y,θ) Gaussian distribution. The general form is given by:

$$P(\hat{x}) = \frac{1}{\sqrt{(2\pi)^n det\ C}} e^{-\frac{1}{2}[(\hat{x}-\hat{X})^t C(\hat{x}-\hat{X})]}$$

where n is the number of dimensions, **C** is the covariance matrix, $\hat{X}$ is the nominal mean vector, and $\hat{x}$ is a vector denoting a particular point. The contours of equal probability of this equation form ellipsoids in n dimensional space that are centered at the mean location $\hat{X}$, and whose axes are only aligned with the Cartesian frame if the covariance matrix **C** is diagonal. The formulas for extracting the principal axes of a 2-D ellipsoid are given below. In the case where we are only interested in the positional error ellipse, **C** is the reduced (2x2) covariance matrix formed from the (3x3) matrix by extracting only the X, Y terms. In this case, the resulting marginal probability density is:

$$P(x,y) = \frac{1}{2\pi\sqrt{\sigma_x^2\sigma_y^2(1-\rho^2)}} e^{-\frac{1}{2(1-\rho^2)}[\frac{x^2}{\sigma_x^2}+\frac{2\rho xy}{\sigma_x\sigma_y}+\frac{y^2}{\sigma_y^2}]} \quad ; \quad C = \begin{pmatrix} \sigma_x^2 & \rho\sigma_x\sigma_y \\ \rho\sigma_x\sigma_y & \sigma_y^2 \end{pmatrix}$$

where ρ is the correlation coefficient for x and y. For example, we might want to check the probability of a camera seeing a particular object given our current locational uncertainty. We decide this question by determining whether the the ellipse corresponding to a given confidence limit is completely contained within the field of view.

For decision making purposes (e.g. the field-of-view-case), it is necessary to determine the explicit equiprobable contours (ellipses or ellipsoids) of the multivariate Gaussian distribution specified by given mean X and covariance $\mathbf{C}_x$ matrices. These ellipses can be used to determine the probability that a given vector will lie within, say, the 90% confidence ellipse. The ellipsoid formula is:

$$(x-X)^t C_x^{-1}(x-X) = k^2 \qquad (A1)$$

where k is a constant chosen for a particular confidence ellipsoid, and x is a point on the ellipsoid boundary. The relationship between k and the probability of a point lying within the ellipsoid specified by k is:

$$\begin{aligned} N &= 1; \qquad P = -\frac{1}{\sqrt{2\pi}} + 2\,erf(k) \\ N &= 2; \qquad P = 1 - e^{\frac{k^2}{2}} \\ N &= 3; \qquad P = -\frac{1}{\sqrt{2\pi}} + 2\,erf(k) - \sqrt{\frac{2}{\pi}} k \cdot \exp^{\frac{k^2}{2}} \end{aligned}$$

where N is the number of dimensions, and erf is the error function.

In the particular case where the two-dimensional (x, y) error ellipse is required given the three dimensional (x, y, θ) covariance matrix, the procedure is as follows. Firstly, produce the 2-D marginal covariance matrix from the 3-D covariance matrix by striking out the row and column of the unwanted variable. That is:

$$C_3(3x3) \Longrightarrow C_2(2x2)$$

The corresponding family of 2-D ellipses is given by Eqn. (A1), and in this case reduces to:

$$Ax^2 + 2Bxy + Cy^2 - k^2 = 0$$

where A, B, C are found from the 2-D covariance matrix and Eqn. (A1). The angle θ the major axis of this ellipse makes with the positive X axis is given by:

$$\theta = \frac{1}{2} \arctan\left(\frac{2B}{A - C}\right); \qquad \theta[\frac{-\pi}{2}, \frac{\pi}{2}]$$

If we define:

$$T = \sqrt{A^2 + C^2 - 2AC + 4B^2}$$

then we find the following lengths:

$$\begin{aligned} half \quad major \quad axis &= \sqrt{\frac{2k^2}{A+C-T}} \\ half \quad minor \quad axis &= \sqrt{\frac{2k^2}{A+C+T}} \end{aligned}$$

As given above, the probability of a point being located inside an ellipse defined by a particular k is given by:

$$P(x, y \in ellipse) = 1 - e^{\frac{-k^2}{2}}; \qquad k^2 = -2\log(1 - Pr)$$

which the following confidence ellipses for different k:

$$\begin{aligned} 50\% &\implies k^2 = 1.386 \\ 90\% &\implies k^2 = 4.605 \end{aligned}$$

Appendix B: Coordinate Frame Mappings

This appendix gives the explicit formulas for mapping the mean and covariance matrix from polar (r, ϕ, θ) representation to an equivalent Cartesian form, where θ is the rotation of the coordinate frame in both the polar and Cartesian case. This mapping is necessary, for example, if a camera expresses the location of an object in polar coordinates and the camera error is also given in polar coordinates, but the result of sensing (or the merged results of many sensings) is required in a Cartesian represenation. The nominal (mean) transformation is given by:

$$\begin{aligned} \hat{x} &= f_1(\hat{r}, \hat{\phi}) = \hat{r}\cos(\hat{\phi}) \\ \hat{y} &= f_2(\hat{r}, \hat{\phi}) = \hat{r}\sin(\hat{\phi}) \\ \hat{\theta} &= f_3 = \hat{\theta} \end{aligned}$$

$\mathbf{R}$, the Jacobian of the transformation, is given by:

$$\mathbf{R} = \begin{pmatrix} \frac{\partial f_1}{\partial r} & \frac{\partial f_1}{\partial \phi} & \frac{\partial f_1}{\partial \theta} \\ \frac{\partial f_2}{\partial r} & \frac{\partial f_2}{\partial \phi} & \frac{\partial f_2}{\partial \theta} \\ \frac{\partial f_3}{\partial r} & \frac{\partial f_3}{\partial \phi} & \frac{\partial f_3}{\partial \theta} \end{pmatrix} = \begin{pmatrix} \frac{x}{r} & -y & 0 \\ \frac{y}{r} & x & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Using a first-order approximation similar to that in Section 3, we get the following Cartesian covariance matrix:

$$\mathbf{C}(x, y, \theta) \cong \mathbf{R} * \mathbf{C}(r, \phi, \theta) * \mathbf{R}^t$$

References

Brooks, R. A. 1985, "Visual Map Making for a Mobile Robot", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Missouri, pp. 824-829.

Brooks, R. A. 1982, "Symbolic Error Analysis and Robot Planning", Int. J. Robotics Res. 1 (4) pp. 29-68.

Chatila, R. and Laumond, J-P. 1985, "Position Referencing and Consistant World Modeling for Mobile Robots", Proc. IEEE Int. Conf. Robotics and Automation, St. Louis, Missouri, pp. 138-145.

Nahi, N. E. 1976, "Estimation Theory and Applications", New York: R. E. Krieger

Paul, R. P. 1981, "Robot Manipulators: Mathematics, Programming and Control", Cambridge, MIT Press

SRI International

FAST ROBOT COLLISION DETECTION USING GRAPHICS HARDWARE

Robotics Laboratory Technical Note

June 1985

By Randall C. Smith
Research Engineer
Robotics Laboratory

SRI Project 7239

The work reported in this paper was supported
by the Air Force Office of Scientific Research
under Contract F49620-84-K-0007

Accepted for presentation at the
Symposium on Robot Control,
Barcelona, Spain, November 6-8, 1985.

SRI International
[illegible] • Menlo Park, CA 94025
[illegible] • TWX [illegible] • Telex [illegible]

CONTENTS

ILLUSTRATIONS

ABSTRACT

Off-line robot programming and simulation systems are emerging based on graphic workstations. These simulate robot motion, but do not usually check for collision of the robot with other objects. A fast method is presented for such detection, making use of existing VLSI graphic hardware. This paper discusses the design of a future robot controller that would use graphic hardware technology to predict potential collisions in real time.

I INTRODUCTION

Robot programming is a task that entails three-dimensional spatial reasoning. The programmer must, for example, create a program that moves a manipulator through a complicated three-dimensional environment so that it does not collide with objects in its path. Advanced graphic hardware addresses the programmer's need to "see" the robot and its environment through graphic simulation, and is helpful in many phases of program development and debugging. However, even graphic-based programming systems should not leave collision detection as an inspection task for the user because:

- The programmer may not have the right "view" to detect the collision visually.
- It may take too long to visually inspect robot trajectories, especially when they are variable (e.g., guided by simulated sensors).

An algorithm for automated collision detection must be fast if it is to serve as a valuable tool for checking interactively defined manipulator trajectories developed with an off-line programming system. In addition, if the robot controller itself is ever to predict and avoid collisions, the algorithm must work fast enough so that the manipulator is not delayed while checking a path.

Accuracy is a requirement for some modes of operation. Collision detection for gross motion planning can be practically implemented by bounding robot components and other objects with simple boxes or spheres [7]; these can be intersected quickly when what is needed is a conservative description of the enclosed shapes, rather than fine detail. Such systems do not work well for fine motion planning, however, for example when the robot is in close proximity to its environment or performing part grasping or part mating operations.

II PREVIOUS WORK

This paper describes a fast method to detect whether two objects intersect, a process often called interference checking. A number of methods for performing interference tests have been described, and can be roughly categorized according to the representations used for the objects.

Interference tests have been reported between simple bounding volumes, such as boxes and spheres [7], between quadratic surfaces [4], or between polyhedra [6], [8]. The work referenced on quadratic surfaces also computed the intersection description. In addition, surfaces composed from bicubic patches can be subdivided into smaller patches approximated by planar regions [2], [1], and the polyhedra intersection tests can be performed.

We will examine the use of "clipping" algorithms from computer graphics to perform fast collision detection, because some of these algorithms can be implemented in hardware. A general outline of collision detection using clipping hardware is given next, followed by a restricted implementation using available hardware. Finally, the concepts are generalized to other possible hardware architectures, concluding with a discussion of the design of a robot controller able to predict collisions (in order to avoid them).

III CLIPPING

Clipping is a fundamental operation in graphics, defined as the process of extracting a portion of a data base. It is used in algorithms for hidden line and hidden surface removal, among others. A basic clipping operation is the intersection of line segments and polygons with a "clipping plane"; only data portions lying on a chosen side of the plane are kept. The chosen side may arbitrarily be defined as the side pointed to by the plane normal. Suppose we wish to clip out surfaces not contained inside a given convex volume, defined by a set of intersecting planes with inward-pointing normals. Clipping a polygon to the convex volume (Figure 1) can be thought of as clipping the polygon to each of the volume's defining planes in turn: At each stage, portions of the polygon lying "outside" the plane are discarded, and the new polygon is passed on for more clipping. Subtleties of clipping algorithms are given in [10].

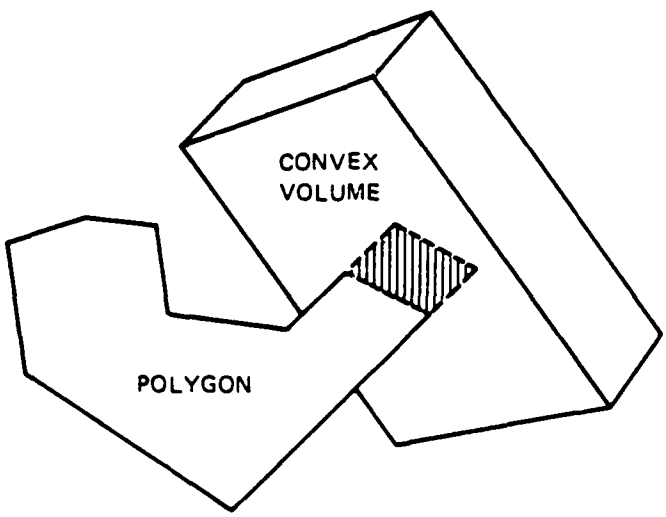


FIGURE 1 CLIPPING TO A CONVEX VOLUME

Algorithms [11] and (later) VLSI chips [3] have been designed to exploit the serial clipping process just described. The VLSI chips are arranged in a

"pipeline" of clipping stages, with each individual "clipper" performing the clipping to its "own" plane. The clippers collectively define a "clipping polyhedron."

Clippers not only determine whether interference occurred, but compute the intersection between the clipping volume and object as well. The extra information will be useful in collision avoidance; i.e., avoiding a collision once it is predicted along the path of the robot. For present purposes, the clipping system need only maintain a vector of bits indicating which, if any, individual planes were intersected.

The most common use of clipping is the extraction of a portion of a world data base to be seen on a two-dimensional screen. Most advanced graphic systems in the future can be expected to have clipping hardware available for that purpose at least. Section V shows how such hardware may be exploited. We will assume the availability of clipping hardware, and capitalize on its speed.

IV GENERAL APPROACH

The method outlined in this paper determines interference between a convex polyhedron (CP) and an arbitrarily constructed obstacle. The convex polyhedron bounds a "test object" to be guarded--presumably a moving component of a manipulator. "Obstacles" are any other modeled objects in the manipulator's workspace. Hardware clippers are loaded with clipping planes that form the boundaries of the convex polyhedron guarding the test object. Obstacle models are then clipped, by hardware, against this clipping polyhedron to determine interference. As in Maruyama's procedure [6] all objects of interest are bounded by circumscribed and inscribed spheres to enable some simple preliminary interference tests. All objects may additionally have enclosing and enclosed convex polyhedra.

The comparisons between the test object and obstacle and their bounding volumes (in order of simplicity) are:

If the outer spheres do not overlap, there is no intersection.
If the inner spheres overlap, there must be an intersection.

If the inner CPs overlap, there must be an intersection.
If the outer CPs do not overlap, there is no intersection.

If the obstacle and outer CP of the test object do not overlap,
there is no intersection.
If the obstacle and inner CP of the test object do overlap,
there must be an intersection.

Otherwise, there MAY BE an intersection between the test object and the obstacle. Further tests would be necessary to decide.

Not all the tests above need be tried. If the obstacle is not very detailed, all of its bounding volumes except the enclosing sphere may be left undefined; testing the obstacle directly will be relatively efficient in that case. The guarded object should at least have an enclosing convex polyhedron defined for it.

Line segments or planar polygons are the primitive elements of each object model tested for interference with the clipping polyhedron. Thus, a general curve is first approximated with a series of short line segments, which are then tested. A surface can be checked against the clipping polyhedron in two ways: by ruling

the surface with a set of curves and intersecting the curves with the polyhedron (see Figure 2), or by approximating the surface as a set of polygons and testing the polygons. If we rule the surface, the curve spacing must be small enough that the clipping polyhedron cannot penetrate the ruled volume to any significant extent without intersecting a curve. Otherwise, the interference would go undetected.

Figure 3 shows how an obstacle model is clipped to the convex polyhedron guarding the test object. The pipeline of clippers is loaded with plane data representing the sides of the clipping polyhedron. Obstacles are broken down into constituent line segments and polygons, whose vertices are transformed into clipping volume coordinates by the 4 x 4 matrix multiplier. A pair of transformed vertices represent a line segment which is passed through the clipping stages, each clipper eliminating any portion of the segment not "inside" its plane. Any data left at the end of the pipeline lies in the interior of the clipping polyhedron. See Appendix A for the procedure details.

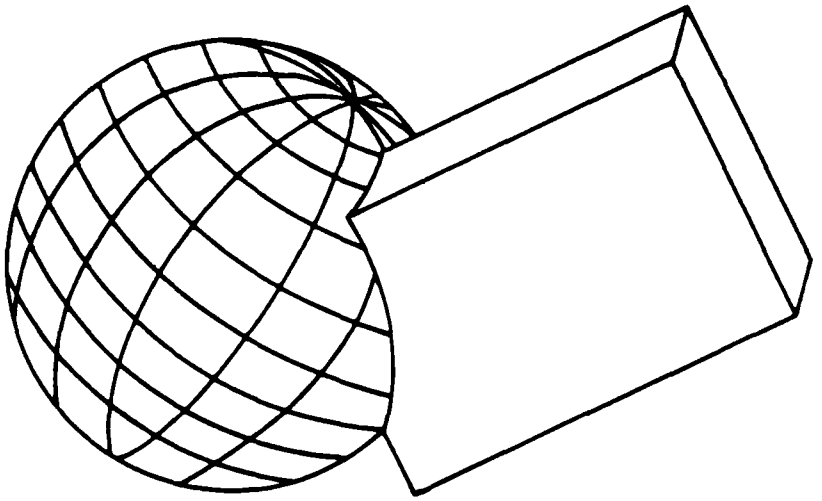


FIGURE 2 INTERFERENCE OF RULED SURFACE WITH TEST POLYHEDRON

A particular implementation of the method is next described which limits the guarding polyhedron for the test object to a parallelopiped (rectangular or skewed) or truncated pyramid (possibly skewed). The limitation arises from the particular organization of the graphic hardware used.

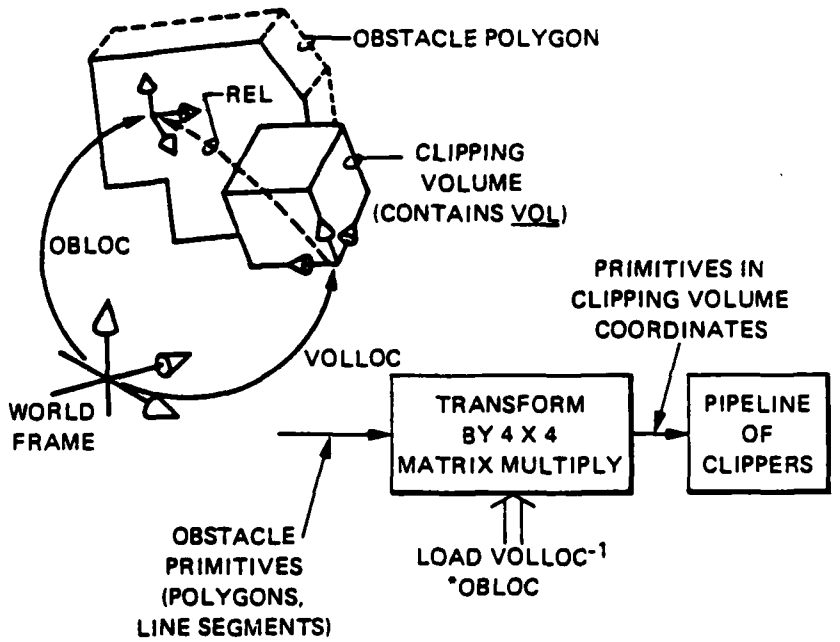


FIGURE 3 CONCEPTUAL MODEL OF AN INTERFERENCE TEST

V IMPLEMENTATION

Collision detection is implementated as part of WORKMATE, an off-line robot programming system under development at SRI based on the Silicon Graphics IRIS 2400 workstation. Objects and manipulators are modeled as collections of convex volumes. Figure 4 shows a Unimation PUMA 560, rendered with shading, whose motions can be computed and drawn approximately six times per second with this level of detail. Collision testing generally takes only a small amount of time, although the time is dependent on the complexity and number of the objects tested. The IRIS transforms and clips primitive coordinates with a hardware pipeline, as depicted in Figure 3, at a rate of about 65,000 coordinates/second, or one coordinate every 16 microseconds. Thus, an obstacle polyhedron with 100 vertices can be checked for collision with the clipping polyhedron in a few milliseconds. In our sub-optimal implementation, an object with 100 vertices is checked for collision with the clipping polyhedron in 7 milliseconds.

The clipping hardware of the IRIS is normally used to limit viewing to a volume of the world specified by the user. The "view volume" is defined by an "eye" peering through a rectangular window. When the eye is placed at infinity, the projection of rays through the window is parallel, and the view volume defined is an infinite parallelopiped; otherwise, the rays passing through the window converge at the eye, defining a semi-infinite pyramid (Figure 5). The volumes are made finite by two planes parallel to the plane of the window, which truncate the view volume at both ends. The IRIS contains six clippers, four to define a rectangular window and two more to define the near and far clipping planes parallel to the window plane.

The class VVOLS is the class of convex polyhedra representable by this standard clipper organization. VVOLS is limited to (potentially skewed) parallelopipeds and truncated pyramids. Thus the outer or inner convex polyhedron guarding any test object must belong to the class VVOLS, in this implementation.

The clippers in the IRIS are preset to clip to a cubical region with diagonally opposite vertices at (1,1,1) and (-1,-1,-1). This arrangement simplifies the plane equations, and thus the clipping calculations performed by the hardware. Before the clippers can clip to the view volume or guarding polyhedron, the volume must first be "normalized" to this cube. A truncated pyramid can be turned into a parallelopiped by the perspective transformation

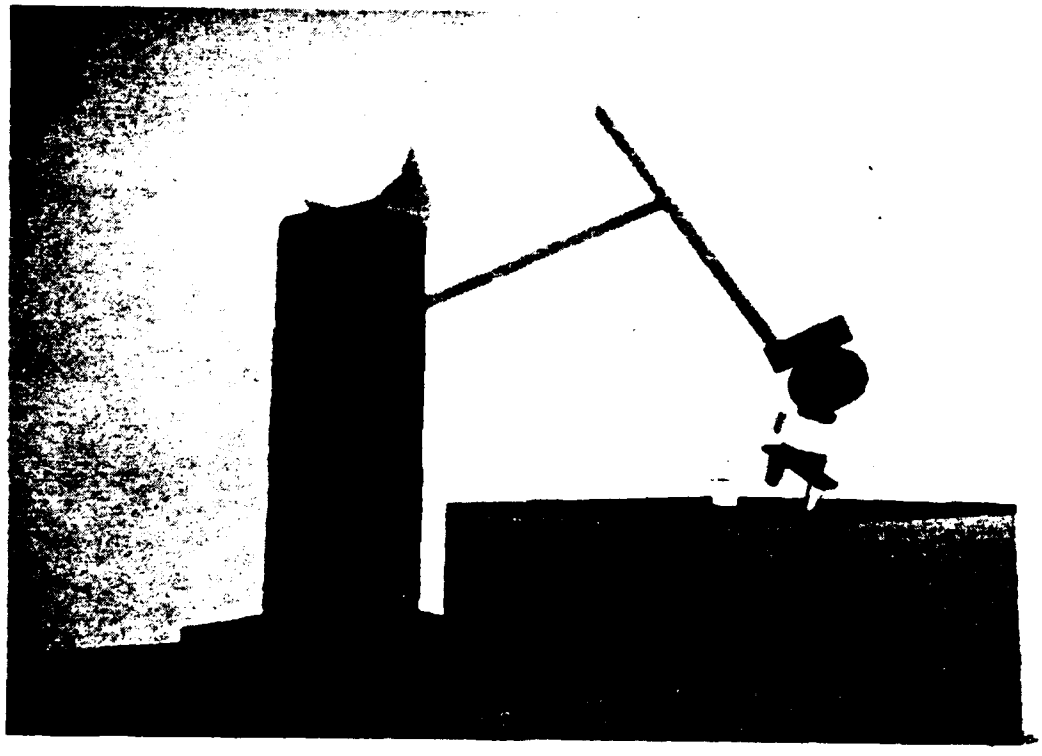


FIGURE 4 AN EXAMPLE SCENE USING WORKMATE

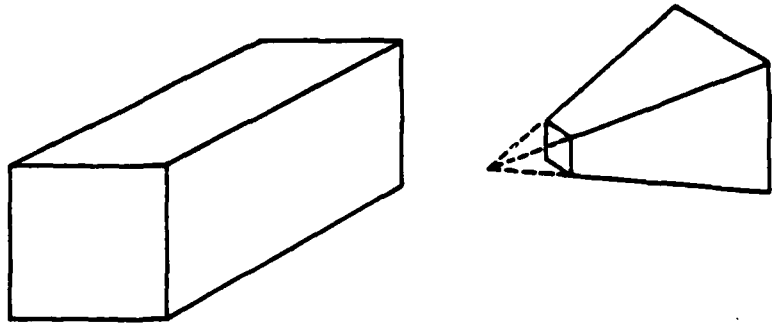


FIGURE 5 VIEW VOLUMES USING A SQUARE WINDOW

(not to be confused with the perspective projection). Skewing can be removed by the shear transformation, producing rectangular parallelopipeds. The result can be scaled to the correct unit dimensions by the scale transformation. All of these operations can be composed into one 4 x 4 matrix using homogenous coordinates [9], [5]. The operation of "normalizing" a view volume to a canonical volume before clipping is described in detail in [5]. The normalizing matrix is computed, a priori, for each different guarding polyhedron to be described, and saved as part of the model.

When a particular test volume (VOL) is selected, the normalization matrix (NORM) for the guarding polyhedron is retrieved, and used to transform the coordinates of obstacle primitives into normalized the coordinates (NVOL) (see Figure 3):

NVOL primitives = NORM * IVOLLOC * OBLOC * OB primitives.

Collision detection then proceeds as previously described.

VI EXTENSIONS

A. Other General Viewing Windows

The implementation was limited to a certain set of polyhedra for bounding volumes because the clipping hardware was set up for a square viewing window. If the clipping VLSI can clip against arbitrary planes, other fixed window arrangements may be useful. For example, octagonal windows define view volumes that are approximations to (potentially skewed) cylinders and cones, under parallel and perspective projection, respectively, as seen in Figure 6. These shapes can be normalized to a unit, 8-sided "cylinder" (plus front and back faces). If the clipping planes marked in the figure can be selectively disabled, the previous class of parallelopipeds and pyramids are included as well, and the normalization matrix is the same. Thus, the hardware could be used for common viewing operations in one mode, and, with the additional four clippers, be able to represent boxes, pyramids, and approximations to cylinders and cones as bounding polyhedra.

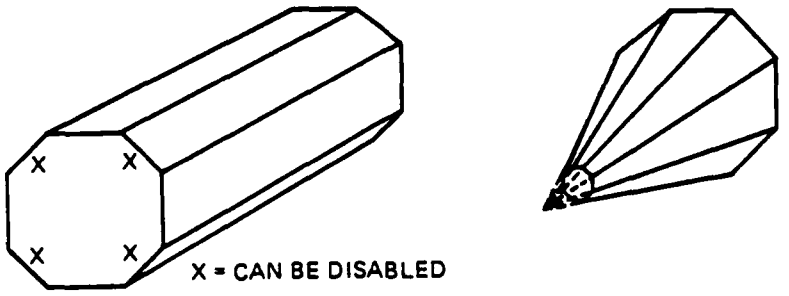


FIGURE 6 VIEW VOLUMES APPROXIMATING CYLINDERS AND CONES

Finally, if a system has n programmable clippers, they can be loaded to represent any n-sided convex volume to be used as a bounding polyhedron.

B. Application to Robot Controller Design

As mentioned at the beginning of the paper, robot trajectories are variable when sensors are used to locate objects for acquisition. It may be impractical for an off-line programming system to determine, a priori, whether robot motions will be collision-free in every circumstance, except in simple environments. To do so analytically might require computing the volume swept out by the manipulator as it moves to every possible location in the range of the object it is acquiring, from every possible starting position, and so on. Monte Carlo simulations may help, but could also be prohibitively time consuming to achieve sufficient reliability.

One solution is to make the collision detection test at "run-time," when the manipulator motion has been determined (perhaps from sensor information). In this situation, of course, the goal is collision prediction: The detection algorithm is applied to points lying ahead of the manipulator on its determined path. A speculative model of a robot controller incorporating collision prediction hardware is illustrated in (Figure 7).

It is presumed that the robot controller can provide information about intermediate destinations along its trajectory, in advance, to the collision prediction subsystem. The length of advance notice is based on the computation speed of the subsystem and a safe reaction interval for the arm, to avoid any impending collision.

The subsystem contains a world model of the arm's surroundings, which may be loaded from an off-line programming system. It also contains a model of the arm subvolumes, 1 to k. Each subvolume is approximated by n planar surfaces defining a convex volume, in its own coordinate system. An associated set of n clippers is preloaded with the set of clipping planes for that convex volume.

The subsystem gets a future point along the current robot trajectory from the controller, and computes new values to be loaded into the matrix multipliers M[i] based on the future locations of the arm subvolumes (see Figure 3 for the computation). The coordinates of the primitive elements for an obstacle are then transformed into each local coordinate system in parallel by the matrix multipliers M[i], and clipped against the modeled arm members. That is, each potential obstacle is checked, in parallel, against the arm subvolumes for intersection. As previously noted, this intersection may take only a few milliseconds for moderately complex obstacles models. The collision prediction subsystem returns

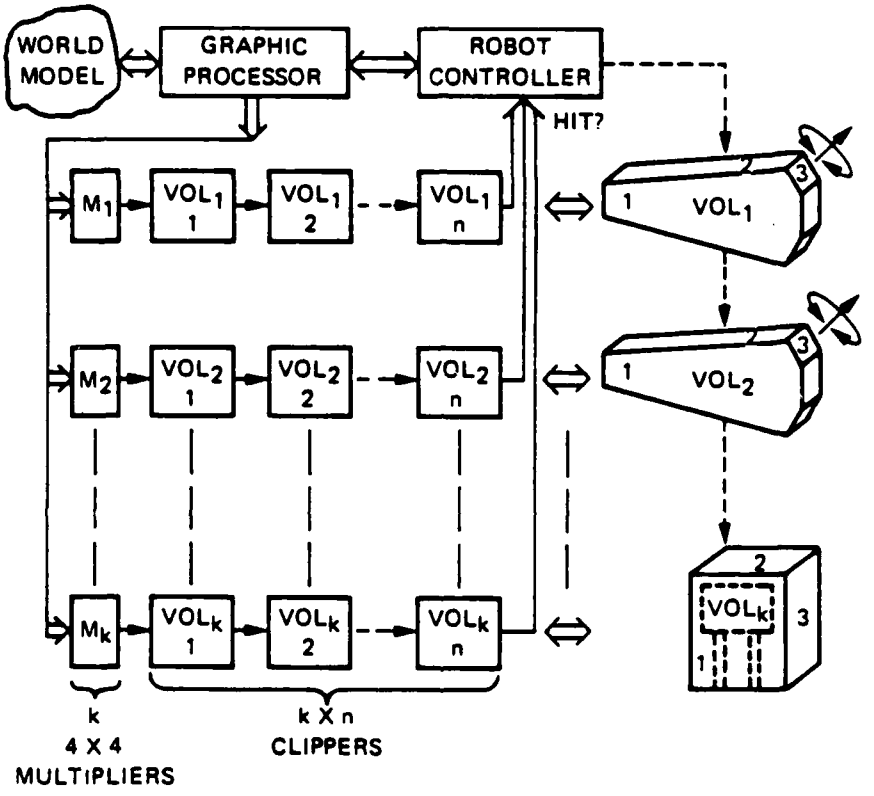


FIGURE 7 ROBOT CONTROLLER WITH COLLISION DETECTION HARDWARE

a status to the robot controller notifying it of any impending collision. An extra matrix multiplier and row of programmable clippers can be used to predict collisions of obstacles with any object held by the manipulator. When an object is grasped, the extra clippers must be programmed to clip to the object's bounding convex polyhedron.

If VLSI clippers and matrix multipliers become abundant and inexpensive because of the graphics market, then collision prediction hardware in the robot controller may become economically feasible.

Appendix A

Collision Detection Using Clipping

Appendix A

Collision Detection Using Clipping

For each object or subobject of interest:

```
CP means convex polyhedron
WLD is the common, world coordinate frame
F(obj) is the coordinate frame of the object obj
RO(obj) is the radius of a circumscribed sphere
RI(obj) is the radius of an inscribed sphere
CS(obj) is the origin of both spheres with respect to F
PO(obj) is an outer, object-enclosing CP defined with respect to F
PI(obj) in an inner, object-enclosed CP defined with respect to F

VOL is the volume to be safeguarded against collision
IVOLLOC is the inverse relation fo F(VOL) to WLD
OB is a particular obstacle from a list of obstacles OL
OBLOC is the relation of F(OB) to WLD
HIT is a vector clipping flags; HIT[k] is ON if the
   kth plane (clipper) intersected OB
```

A description of the method is as follows:

```
Choose VOL and OL.
Clear the HIT vector.

For each obstacle OB in OL

NEXTOB: if no more obstacles     return NO INTERSECTION.
1.      [Compute the relation of F(OB) to F(VOL):  ]
                REL = IVOLLOC * OBLOC.

2.      [Compute distance between bounding spheres of VOL and OB.]
        D = dist(CS(VOL), REL * CS(OB))

3.      [If spheres don't overlap, no intersection with this OB.]
        if D > RO(VOL) + RO(OB)           goto NEXTOB.

4.      [If inside spheres overlap, VOL and OB must intersect.]
        if both RI are defined then
          if D < RI(VOL) + RI(OB)         return INTERSECTED.

5.      if COLLIDE( PO(OB), PO(VOL), REL ) = no
                                          goto NEXTOB.
6.      if COLLIDE( PI(OB), PI(VOL), REL ) = yes
                                          return INTERSECTED.
7.      if COLLIDE( OB, PI(VOL), REL ) = yes
                                          return INTERSECTED.
8.      if COLLIDE( OB, PO(VOL), REL ) = no
                                          goto NEXTOB.
        else                              return MAYBE INTERSECTED.


COLLIDE(OBJ, CLIPVOL, RELATION)
        If OBJ or CLIPVOL not defined     return ignore.
        Load clippers with CLIPVOL plane data.
        For each primitive P of OBJ
           Convert each vertex V of P to F(CLIPVOL) coordinates
                 by V' = RELATION * V
           Clip P' against CLIPVOL
           If any HIT[k] are ON,
                clear HIT.                return yes.
        [No more primitives]              return no.
```

If the routine returns "maybe," a collision may have occurred, but the test results are not conclusive. Further testing could be implemented by subdividing the volume between PO(VOL) and PI(VOL) into convex subvolumes, and clipping OB against them. The subdivision is performed just once, when the model of VOL is defined. Our implementation, however, does not carry out these second-level tests.

References

1. Blinn J. F., L. Carpenter, J. Lane, and T. Whitted, "Scan Line Methods for Displaying Parametrically Defined Surfaces," *Communications of the ACM*, Vol. 23, No. 1 (January 1980) 23-34.

2. Clark, J. H., "Parametric Curves, Surfaces, and Volumes in Computer Graphics and Computer-Aided Geometric Design," NASA Ames Research Center (1978).

3. Clark J. H., "A VLSI Geometry Processor for Graphics," *IEEE Computer*, Vol. 12, No. 7 (July 1980).

4. Mathematical Applications Group, Inc., "3-D Simulated Graphics," *Datamation*, Vol. 1 (February 1968).

5. Foley, J. D., and A, Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, Massachussets, 1982.

6. Maruyama, K., "A Procedure to Determine Intersections Between Polyhedral Objects," *Int. J. Comp. Inf. Sci.*, Vol. 1, No. 3 (1972) 255-266.

7. Myers, J.K., "RCODE: The Robotic Simulator with Collision Detection," Robotics Laboratory Technical Note, SRI International, Menlo Park, California (December 1984).

8. Newell, M. E., "The Utilizations of Procedure Models in Digital Image Synthesis," UTEC-CSc-76-218, University of Utah, Computer Science Department (Summer 1975).

9. Paul, R. P., *Robot Manipulators: Mathematics, Programming, and Control*, MIT Press, Cambridge, Massachussets, 1982.

10. Rogers, D. F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, 1985.

11. Sutherland, I. E., and G. W. Hodgman, "Reentrant Polygon Clipping," *Communications of the ACM*, Vol. 17, No. 1 (January 1974) 32-42.

END

DTIC

7-86